EXHIBIT C

System.Firewall.P licy.PolicyAction

```
namespace System.Firewall.Policy
{
    public abstract class PolicyAction : PolicyObject
    {
    // Properties
    public LoggingConfig LoggingConfig { get { } }
    // Abstract methods
        public abstract bool IsCompatible(PolicyCondition condition);
    }
}
```

PolicyAction is an abstract class that serves as the root class for all specific action classes. Whether to log the packet is implied by specifying a non-empty logging configure object.

Methods

Method	
Name	IsCompatible
Return Type	Bool
Description	Return true if this action can take place at the same layer as the condition. Otherwise return false.
Parameters	Condition - The condition against which this action will be checked.

System.Firewall.P licy.FilterAction

```
namespace System.Firewall.Policy

{

      public class FilterAction : PolicyAction

      {

            public enum FilterActionType { Permit, Deny, LogOnly }

                  private FilterAction(FilterActionType actionType);

            public FilterAction(FilterActionType actionType, LoggingConfig
logging);

            public static readonly FilterAction Permit = new FilterAction(Permit);

      public static readonly FilterAction Deny = new FilterAction(Deny);


    public override bool IsCompatible(PolicyCondition condition) { return true;}

      }

};
```

FilterAction models the following action scenarios:

- Permit: Allow packets that match the associating PolicyCondition object.

```
    PolicyAction action = FilterAction.Permit;
```

- Deny: Drop packets that match the associating PolicyCondition object.

```
    PolicyAction action = FilterAction.Deny;
```

- Permit and Log: Allow and log packets that match the associating PolicyCondition object.

```
    new

FilterAction(FilterAction.FilterActionType.Permit,

logging);
```

- Deny and Log: Drop and log packets that match the associating PolicyCondition object.

```
      new FilterAction(FilterAction.FilterActionType.Deny,
logging);
```

- Log Only: Log packets that match the associating PolicyCondition object.

```
      new
FilterAction(FilterAction.FilterActionType.LogOnly,
logging);
```

Please note that FilterAction can be associated with any PolicyCondition as it is required that all layers in the firewall platform will at least support Permit, Deny and Log actions.

System.Firewall.P licy.Instantiati nTemplate

```
namespace System.Firewall.Policy

{

      public abstract class InstantiationTemplate : PolicyRule

      {

        protected InstantiationTemplate(bool isClientInstantiation);

              public bool IsClientInstantiation { get { } };

      }

      public class TransportTemplate : InstantiationTemplate

      {

        public TransportTemplate(TransportLayer layer, FitlerAction action);

        public TransportTemplate(TransportLayer layer, FilterAction action,

             IPAddressValue remoteAddr, UInt16Value protocol, UInt16Value

             remotePort);

              public TransportCondition Condition { get { } }

              public FilterAction Action { get { } }

      }


      public class IPSecTemplate : InstantiationTemplate

      {

        public IPSecTemplate(IPSecAction action);

        public IPSecTemplate(IPSecAction action, IPAddressValue remoteAddress,

         UInt16Value protocol, UInt16Value remotePort);

        public TransportCondition Condition { get { } }

        public IPSecAction Action { get { } }

      }

      public class IPSecAuthorizationTemplate : InstantiationTemplate

      {

        public IPSecAuthorizationTemplate(RemoteIdentityValue remoteID,

             FilterAction action);

        public IPSecAuthorizationTemplate(RemoteIdentityValue remoteID, UInt16Value
        protocol, UInt16Value remotePort, FilterAction action);

        public RemoteIdentityValue RemoteID { get { } }

        public FilterAction Action { get { } }

    }

}
```

Instantiation templates can be any of the following templates:

- **TransportTemplate**: the template to be instantiated at the transport layer either inbound or outbound.
- **IPSecLayer**: the IPsec template to be instantiated at IPSec layer.
- **IPSec Authorization rule template**: the authorization template to be instantiated at the IPSec authorization layer.

Each of the above classes provide two constructors. The first one is to be instantiated when the associated application rule matched to perform client instantiation. When client instantiation takes place, the full 5-tuple is available to instantiate the template. On the other hand, if it is not a client instantiation, only local 3-tuple, i.e. local address, protocol, and local port, available. This is what the second constructor will be used for. So implicitly, the first constructor set the isClientInstantiation flag in the base InstantiationTemplate class to be true while the second one set it to false. The client instantiation templates are instantiated only when the full 5-tuple is available while the server instantiation templates only at the time when the local 3-tuple is available.

System.Firewall.P licy.Applicati nAction

```
namespace System.Firewall.Policy
{
        public class ApplicationAction : PolicyAction
        {
                public enum ApplicationActionType
                {
                        Permit, Deny, Ask, LogOnly
                }
                        private ApplicationAction(ApplicationActionType actionType);
                public ApplicationAction(ApplicationActionType actionType,
                LoggingConfig logging, InstantiateTemplateCollection templates);
                public static readonly ApplicationAction Permit = new
        ApplicationAction(Permit);
        public static readonly ApplicationAction Deny = new ApplicationAction(Deny);
        public static readonly ApplicationAction Ask = new ApplicationAction(Ask);
                public InstantiateTemplateCollection InstantiationTemplates { get {   }
set { } }
        public IPSecProposal IPSecProposal { get { } set { } };
        public bool IsAutoInstantiationEnabled { get { } set { } }
        public override bool IsCompatible(PolicyCondition condition);
        }
};
```

The possible ApplicationAction scenarios are as following:

- **Permit**: Allow packets that match the associating ApplicationCondition object.
- **Deny**: Drop packets that match the associating ApplicationCondition object.
- **Ask**: Ask for users' decisions when packets match the associating ApplicationCondition object.

- **Log Only**: Log packets that match the associating ApplicationCondition object.

Methods

Method	
Name	GetTemplates
Return Type	InstantiateTemplateCollection
Description	Return the list of instantiate templates that will be created when this application action takes place.
Parameters	Condition - The condition against which this action will be checked.

System.Firewall.P licy.CalloutAction

```
namespace System.Firewall.Policy

{

      public abstract class CalloutAction : PolicyAction

      {

            // Constructors

            protected CalloutAction (Callout obj, CalloutContext cxt);

            // Properties

            public Callout CalloutModule { get { } }

            public CalloutContext Context { get { } set { } }

      public override bool IsCompatible(PolicyCondition condition);

      }

};
```

CalloutAction models the callout extensions that the platform provides. When associating conditions are matched, the callout action specifies the callout extension that the platform needs to invoke. It is used as an extension mechanism to provide additional security functionalities like intrusion detection, parental control etc.

Property	
Name	CalloutModeul
Description	The callout module to be invoked when the associating condition is matched.
Access	Read Only

Property	
Name	CalloutContext
Descripti n	The callout specific context information

	to be passed to the callout module when it is invoked.
Access	Read/Write

System.Firewall.Policy.IPSecAction

```
namespace System.Firewall.Policy

{

        public class IPSecProposal : CalloutContext

        {

                public IPSecProposal();

                public bool IsPFSRequired { get { } set { } }

                // A flag indicating if can send in clear (soft SA) when key
negotiation fails.

                public bool IsAuthenticationRequired { get { } set { } }

                public bool IsNATTraversalEnabled { get { } set { } }

                public HashAlgorithm AHTransform { get { } set { } }

                public HashAlgorithm ESPIntegrityTransform { get { } set { } }

                public CiperAlgorithm ESPCiperTransform { get { } set { } }

                public uint32 MaxLifetimeSeconds { get { } set { } }

                public uint32 MaxLifetimeKilobytes { get { } set { } }

        }

        public class IPSecAction : CalloutAction

        {

                // Constructors

                // call base constructor with the Callout object for IPSec callout
module and

                // null for context.

                Public IPSecAction();

                Public IPSecAction(IPSecProposal ctx);

                                public IPSecProposal Context { get { } set { } }

        }

};
```

IPSecAction triggers the IPSec callout to set a security context in the matching packets so that the packets will be further process by the IPSec driver. It also specifies the actual IPSec configure parameters for securing network traffic, including AH or ESP or both and their corresponding transform settings.

System.Firewall.P licy.IKEAction

```
namespace System.Firewall.Policy
{
         pulbic enum IKEAuthenticationType
         {
      PresharedKey = 1,
      Kerberos = 2,
      Passport = 3,
      Certificate = 4
         }
            public class IKEAuthenticationMethod : PolicyObject
      {
           protected IKEAuthenticationMethod(IKEAuthenticationType authType);
            public static readonly IKEAuthenticationMethod PresharedKey = new
IKEAuthenticationMethod(PresharedKey);
            public static readonly IKEAuthenticationMethod Kerberos = new
IKEAuthenticatinMethod(Kerberos);
           public static readonly IKEAuthenticationMethod Passport = new
IKEAuthenticatinMethod(Passport);
      }
      public class CertificateAuthenticationMethod : IKEAuthenticationMethod
        {
        public CertificateAutenticationMethod();
        public X509CertificateCollection RootCertificates { get { } set { } }
         }
      public enum CipherAlgorithm
      {
        None,
        DES,
        3DES
      }
```

```
    public enum HashAlgorithm
  {
    None,
    MD5,
    SHA
  }
  public class IKEProposal : PolicyObject
  {
      public IKEProposal(CipherAlgorithm ciper, HashAlgorithm hash);
    // . Predefine high, medium, and low proposals as static variables.
    public CipherAlgorithm CiperAlgorithm { get { } }
    public HashAlgorithm HashAlgorithm { get { } }
      public uint32 MaxLifetimeSeconds { get { } set { } }
      public uint32 MaxLifetimeKilobytes { get { } set { } }
    public uint32 MaxQuickModeNumber { get { } set { } }
  }
  public class IKEAction : PolicyAction
  {
      public IKEAction(IKEAuthenticationMethod method);
       // Authentication Method: Pre-shared key, Kerberos, certificate
       // (certificates for outbound, certificates for inbound)
       public IKEAuthenticationMethod AuthenticationMethod { get { } }
       // Proposal for algorithms etc.
       public IKEProposal Proposal { get { } set { } }
  }
};
```

IKEAction defines the authentication methods for performing IKE key negotiation protocol, which can be either pre-shared key, Kerberos or certificates, and also proposals for authentication algorithms.

System.Firewall.P licy.KeyingM duleActi n

```
namespace System.Firewall.Policy

{

    public class KeyingModuleAction : PolicyAction

    {

        public enum KeyingModule

        {

            IKE,

            Mamie

        }

        private KeyingModuelAction(KeyingModule module);

        public KeyingModuleAction(KeyingModuleCollection modules);

        public static readonly IKE = new KeyingModuleAction(IKE);

        public static readonly Mamie = new KeyingModuleAction(Mamie);

        public KeyingModeuleCollection GetKeyingModules();

    }

};
```

KeyingModuleAction selects the specified keying module to perform key negotiation exchanges. When more than one is specified, each of the listed keying modules will be tried in order until one of them succeeds or all have failed.

Methods

Method	
Name	GetKeyingModules
Return Type	KeyingModuleCollection
Description	Return one or more keying modules that may be invoked when this action is taken place. If more than one keying modules are listed, they will be tried in the order as specified until one of them succeeds or all fail.

Parameters	None.